

# Logic Characterization of Floyd Languages

Violetta Lonati<sup>1</sup>, Dino Mandrioli<sup>2</sup>, Matteo Pradella<sup>2</sup>

<sup>1</sup> DSI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy  
lonati@dsi.unimi.it

<sup>2</sup> DEI - Politecnico di Milano, via Ponzio 34/5, Milano, Italy  
{dino.mandrioli, matteo.pradella}@polimi.it

**Abstract.** Floyd languages (FL), alias Operator Precedence Languages, have recently received renewed attention thanks to their closure properties and local parsability which allow one to apply automatic verification techniques (e.g. model checking) and parallel and incremental parsing. They properly include various other classes, noticeably Visual Pushdown languages. In this paper we provide a characterization of FL in terms a monadic second order logic (MSO), in the same style as Büchi's one for regular languages. We prove the equivalence between automata recognizing FL and the MSO formalization.

**Keywords:** Operator precedence languages, Deterministic Context-Free languages, Monadic Second-Order Logic, Pushdown automata.

## 1 Introduction

Floyd languages (FL), as we recently renamed Operator Precedence Languages after their inventor, were originally introduced to support deterministic parsing of programming and other artificial languages: by taking inspiration from the structure of arithmetic expressions, which gives precedence to multiplicative operations w.r.t. additive ones, Robert Floyd defined an operator precedence matrix (OPM) associated with a context-free (operator) grammar. When the OPM is free of conflicts it is easy to build a deterministic shift-reduce algorithm that associates any language sentence with a unique syntax tree [1]. FL and related grammars (FG) were also studied with different motivations, such as grammar inference. This lead to discover interesting closure properties that are not enjoyed by more general context-free (CF) languages [2]. After these initial results the interest in FL properties decayed for several decades, probably due to the advent of more expressive grammars, such as LR ones [3] which also allow for efficient deterministic parsing.

Recently, however, we revitalized our interest in FL on the basis of two rather unexpected remarks. First, and rather occasionally, we noted that a newer class of CF deterministic languages, namely Visual Pushdown Languages (VPL) -and other connected families [4,5,6]- are a proper subclass of FL. VPL have been introduced and investigated [7] with the main motivation to extend to them the same or similar automatic analysis techniques -noticeably, model checking- that have been so successful for regular languages; their major features which made them quite successful in the literature are that: despite being recognized by infinite state machines -a specialized class of pushdown automata- they enjoy practically all closure properties exhibited by regular

languages; they can be defined by a suitable logic formalism that extends in a fairly natural way the classical Monadic Second Order (MSO) logic characterization introduced by Büchi for finite state automata [8]. These features, paired with the decidability of the emptiness problem shared by all CF languages, makes them amenable for the application of typical model checking techniques. When we realized that VPL are subclass of FL characterized by a well-precise “shape” of OPM we also investigated other closure properties that were not yet known: by joining old results of decades ago [9] with new ones [2], it turns out the FL enjoy the same closure properties w.r.t. main operations such as Boolean ones, concatenation, Kleene \*, etc. as regular languages and VPL. Thus, FL too are amenable for a significant extension of model checking techniques.

A second major motivation that renewed our interest in FL -which, however, has a lesser impact on the present research- is their locality principle, which makes them much better suited than other deterministic CF languages for parallel and incremental (parsing) techniques: unlike more general languages, in fact, the parsing of a substring  $w$  of a string  $x$  can be carried over independently of the “context” of  $w$  within  $x$ ; we feel that in the era of multicore machines the minor loss in expressive power of FG w.r.t. say, LR ones, is far compensated by the gain of efficiency in -possibly incremental analysis- that can be obtained by exploiting parallelism [10].

In our path of “rediscovering FL and their properties”, we also filled up a fairly surprising hole in previous literature, namely the lack of an automata family that perfectly matches FG in terms of generative power: Floyd Automata (FA) are reported in [11] and, with more details and precision, in [12].

In this paper we provide the “last tile of the puzzle”, i.e., a complete characterization of FL in terms of a suitable MSO, so that, as well as with regular languages and VPL, one can, for instance, state a language property by means of a MSO formula; then automatically verify whether a given FA accepts a language that enjoys that property. Our new MSO logic is certainly inspired by the original [8] approach, as well as the technique to automatically derive a FA from a given formula; as it happened also with other previous “extensions” of properties and techniques to the FL family, however, we had to face some new technical difficulties which sharply departed from the original approaches of both regular and VPL [8], [13]. In this case the main difference between finite state automata and VPA on one side and FA on the other one is that the former ones are real-time machines -i.e. read an input character at any move, whereas FA are not; thus, properties expressed in terms of character positions cannot exploit the fact that to any position it corresponds one and only one state of the automaton. In some sense the logic formalization of a FL must encode the corresponding parsing algorithm which is far from the trivial one of regular and VPL whose strings have a shape isomorphic to the corresponding syntax tree.

The paper is structured as follows: Section 2 provides the necessary background about FL and their automata. Section 3 defines a MSO over strings and provides two symmetric constructions to derive an equivalent FA from a MSO formula and conversely. Section 4 offers some conclusion and hints for future work.

## 2 Preliminaries

FL are normally defined through their generating grammars [1,14]; in this paper, however, we characterize them through their accepting automata [12,11] which are the natural way to state equivalence properties with logic characterization. Nevertheless we assume some familiarity with classical language theory concepts such as context-free grammar, parsing, shift-reduce algorithm, syntax tree [3].

Let  $\Sigma = \{a_1, \dots, a_n\}$  be an alphabet. The empty string is denoted  $\epsilon$ . We use a special symbol  $\#$  not in  $\Sigma$  to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the look-back and look-ahead of one character to determine the next parsing action [3].

**Definition 1.** An operator precedence matrix (OPM)  $M$  over an alphabet  $\Sigma$  is a partial function  $(\Sigma \cup \{\#\})^2 \rightarrow \{<, \doteq, >\}$ , that with each ordered pair  $(a, b)$  associates the OP relation  $M_{a,b}$  holding between  $a$  and  $b$ . We call the pair  $(\Sigma, M)$  an operator precedence alphabet (OP). Relations  $<, \doteq, >$ , are named yields precedence, equal in precedence, takes precedence, respectively. By convention, the initial  $\#$  can only yield precedence, and other symbols can only take precedence on the ending  $\#$ .

If  $M_{a,b} = \circ$ , where  $\circ \in \{<, \doteq, >\}$ , we write  $a \circ b$ . For  $u, v \in \Sigma^*$  we write  $u \circ v$  if  $u = xa$  and  $v = by$  with  $a \circ b$ .  $M$  is complete if  $M_{a,b}$  is defined for every  $a$  and  $b$  in  $\Sigma$ . Moreover in the following we assume that  $M$  is acyclic, which means that  $c_1 \doteq c_2 \doteq \dots \doteq c_k \doteq c_1$  does not hold for any  $c_1, c_2, \dots, c_k \in \Sigma, k \geq 1$ . See [9,2,12] for a discussion on this hypothesis.

**Definition 2.** A nondeterministic Floyd automaton (FA) is a tuple  $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$  where:

- $(\Sigma, M)$  is a precedence alphabet,
- $Q$  is a set of states (disjoint from  $\Sigma$ ),
- $I, F \subseteq Q$  are sets of initial and final states, respectively,
- $\delta : Q \times (\Sigma \cup Q) \rightarrow 2^Q$  is the transition function.

The transition function is the union of two disjoint functions:

$$\delta_{\text{push}} : Q \times \Sigma \rightarrow 2^Q \quad \delta_{\text{flush}} : Q \times Q \rightarrow 2^Q$$

A nondeterministic FA can be represented by a graph with  $Q$  as the set of vertices and  $\Sigma \cup Q$  as the set of edge labelings: there is an edge from state  $q$  to state  $p$  labelled by  $a \in \Sigma$  if and only if  $p \in \delta_{\text{push}}(q, a)$  and there is an edge from state  $q$  to state  $p$  labelled by  $r \in Q$  if and only if  $p \in \delta_{\text{flush}}(q, r)$ . To distinguish flush transitions from push transitions we denote the former ones by a double arrow.

To define the semantics of the automaton, we introduce some notations. We use letters  $p, q, p_i, q_i, \dots$  for states in  $Q$  and we set  $\Sigma' = \{a' \mid a \in \Sigma\}$ ; symbols in  $\Sigma'$  are called marked symbols. Let  $\Gamma = (\Sigma \cup \Sigma' \cup \{\#\}) \times Q$ ; we denote symbols in  $\Gamma$  as  $[a q]$ ,  $[a' q]$ , or  $[\# q]$ , respectively. We set  $\text{smb}([a q]) = \text{smb}([a' q]) = a$ ,  $\text{smb}([\# q]) = \#$ , and  $\text{st}([a q]) = \text{st}([a' q]) = \text{st}([\# q]) = q$ .

A *configuration* of a FA is any pair  $C = \langle B_1 B_2 \dots B_n, a_1 a_2 \dots a_m \rangle$ , where  $B_i \in \Gamma$  and  $a_i \in \Sigma \cup \{\#\}$ . The first component represents the contents of the stack, while the second component is the part of input still to be read.

A computation is a finite sequence of moves  $C \vdash C_1$ ; there are three kinds of moves, depending on the precedence relation between  $smb(B_n)$  and  $a_1$ :

**(push)** if  $smb(B_n) \doteq a_1$  then  $C_1 = \langle B_1 \dots B_n[a_1 q], a_2 \dots a_m \rangle$ , with  $q \in \delta_{push}(st(B_n), a_1)$ ;

**(mark)** if  $smb(B_n) \triangleleft a_1$  then  $C_1 = \langle B_1 \dots B_n[a_1' q], a_2 \dots a_m \rangle$ , with  $q \in \delta_{push}(st(B_n), a_1)$ ;

**(flush)** if  $smb(B_n) \triangleright a_1$  then let  $i$  the greatest index such that  $smb(B_i) \in \Sigma'$ .

$$C_1 = \langle B_1 \dots B_{i-2}[smb(B_{i-1}) q], a_1 a_2 \dots a_m \rangle, \text{ with } q \in \delta_{flush}(st(B_n), st(B_{i-1})).$$

Finally, we say that a configuration  $[\# q_I]$  is *starting* if  $q_I \in I$  and a configuration  $[\# q_F]$  is *accepting* if  $q_F \in F$ . The language accepted by the automaton is defined as:

$$L(\mathcal{A}) = \left\{ x \mid \langle [\# q_I], x\# \rangle \vdash^* \langle [\# q_F], \# \rangle, q_I \in I, q_F \in F \right\}.$$

Notice that transition function  $\delta_{push}$  is used to perform both push and mark moves. To distinguish them, in the graphical representation of a FA we will use a solid arrow to denote mark moves in the state diagram.

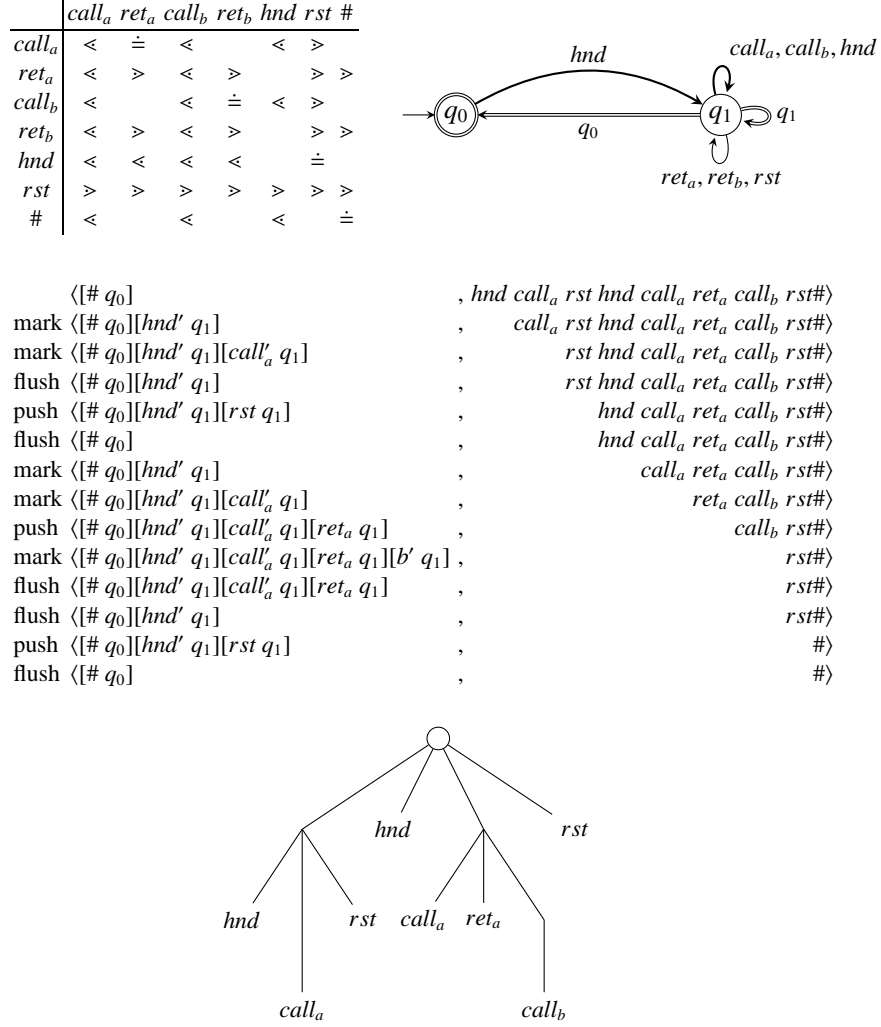
The deterministic version of FA is defined along the usual lines.

**Definition 3.** A FA is *deterministic* if  $I$  is a singleton, and the ranges of  $\delta_{push}$  and  $\delta_{flush}$  are both  $Q$  rather than  $2^Q$ .

In [12] we proved in a constructive way that nondeterministic FA have the same expressive power as the deterministic ones and both are equivalent to the original Floyd grammars.

*Example 1.* We define here the stack management of a simple programming language that is able to handle nested exceptions. For simplicity, there are only two procedures, called  $a$  and  $b$ . Calls and returns are denoted by  $call_a, call_b, ret_a, ret_b$ , respectively. During execution, it is possible to install an exception handler  $hnd$ . The last signal that we use is  $rst$ , that is issued when an exception occur, or after a correct execution to uninstall the handler. With a  $rst$  the stack is “flushed”, restoring the state right before the last  $hnd$ . Every  $hnd$  not installed during the execution of a procedure is managed by the OS. We require also that procedures are called in an environment controlled by the OS, hence calls must always be performed between a  $hnd/rst$  pair (in other words, we do not accept *top-level* calls). The automaton modeling the above behavior is presented in Figure 1.

Incidentally, notice that such a language is not a VPL but somewhat extends their rationale: in fact, whereas VPL allow for unmatched parentheses only at the beginning of a sentence (for returns) or at the end (for calls), in this language we can have unmatched  $call_a, call_b, ret_a, ret_b$  within a pair  $hnd, rst$ .



**Fig. 1.** Precedence matrix, automaton, example run, and corresponding tree of Example 1.

**Definition 4.** A simple chain is a string  $c_0c_1c_2 \dots c_\ell c_{\ell+1}$ , written as  ${}^{c_0}[c_1c_2 \dots c_\ell]^{c_{\ell+1}}$ , such that:  $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$ ,  $c_i \in \Sigma$  for every  $i = 1, 2, \dots, \ell$ , and  $c_0 < c_1 \doteq c_2 \dots c_{\ell-1} \doteq c_\ell > c_{\ell+1}$ . A composed chain is a string  $c_0s_0c_1s_1c_2 \dots c_\ell s_\ell c_{\ell+1}$ , where  ${}^{c_0}[c_1c_2 \dots c_\ell]^{c_{\ell+1}}$  is a simple chain, and  $s_i \in \Sigma^*$  is the empty string or is such that  ${}^{c_i}[s_i]^{c_{i+1}}$  is a chain (simple or composed), for every  $i = 0, 1, \dots, \ell$ . Such a composed chain will be written as  ${}^{c_0}[s_0c_1s_1c_2 \dots c_\ell s_\ell]^{c_{\ell+1}}$ .

A string  $s \in \Sigma^*$  is compatible with the OPM  $M$  if  $\#[s]^\#$  is a chain.

**Definition 5.** Let  $\mathcal{A}$  be a Floyd automaton. We call a support for the simple chain  ${}^{c_0}[c_1c_2 \dots c_\ell]^{c_{\ell+1}}$  any path in  $\mathcal{A}$  of the form

$$q_0 \xrightarrow{c_1} q_1 \longrightarrow \dots \longrightarrow q_{\ell-1} \xrightarrow{c_\ell} q_\ell \xRightarrow{q_0} q_{\ell+1} \quad (1)$$

Notice that the label of the last (and only) flush is exactly  $q_0$ , i.e. the first state of the path; this flush is executed because of relation  $c_\ell > c_{\ell+1}$ .

We call a support for the composed chain  ${}^{c_0}[s_0c_1s_1c_2 \dots c_\ell s_\ell]^{c_{\ell+1}}$  any path in  $\mathcal{A}$  of the form

$$q_0 \xrightarrow{s_0} q'_0 \xrightarrow{c_1} q_1 \xrightarrow{s_1} q'_1 \xrightarrow{c_2} \dots \xrightarrow{c_\ell} q_\ell \xrightarrow{s_\ell} q'_\ell \xRightarrow{q'_0} q_{\ell+1} \quad (2)$$

where, for every  $i = 0, 1, \dots, \ell$ :

- if  $s_i \neq \epsilon$ , then  $q_i \xrightarrow{s_i} q'_i$  is a support for the chain  ${}^{c_i}[s_i]^{c_{i+1}}$ , i.e., it can be decomposed as  $q_i \xrightarrow{s_i} q''_i \xRightarrow{q_i} q'_i$ .
- if  $s_i = \epsilon$ , then  $q'_i = q_i$ .

Notice that the label of the last flush is exactly  $q'_0$ .

The chains fully determine the structure of the parsing of any automaton over  $(\Sigma, M)$ . Indeed, if the automaton performs the computation

$$\langle [a q_0], sb \rangle^* \langle [a q], b \rangle.$$

then  ${}^a[s]^b$  is necessarily a chain over  $(\Sigma, M)$  and there exists a support like (2) with  $s = s_0c_1 \dots c_\ell s_\ell$  and  $q_{\ell+1} = q$ .

Furthermore, the above computation corresponds to the parsing by the automaton of the string  $s_0c_1 \dots c_\ell s_\ell$  within the context  $a, b$ . Notice that such context contains all information needed to build the subtree whose frontier is that string. This is a distinguishing feature of FL, not shared by other deterministic languages: we call it the *locality principle* of Floyd languages.

*Example 2.* With reference to the tree in Figure 1, the parsing of substring  $hnd call_a rst hnd$  is given by computation

$$\langle [\# q_0], hnd call_a rst hnd \rangle^* \langle [\# q_0], hnd \rangle$$

which corresponds to support  $q_0 \xrightarrow{hnd} q_1 \xrightarrow{call_a} q_1 \xRightarrow{q_1} q_1 \xrightarrow{rst} q_1 \xRightarrow{q_0} q_0$  of chain  $\#[hnd call_a rst]^{hnd}$ .

**Definition 6.** Given the OP alphabet  $(\Sigma, M)$ , let us consider the FA  $\mathcal{A}(\Sigma, M) = \langle \Sigma, M, \{q\}, \{q\}, \{q\}, \delta_{\max} \rangle$  where  $\delta_{\max}(q, q) = q$ , and  $\delta_{\max}(q, c) = q, \forall c \in \Sigma$ . We call  $\mathcal{A}(\Sigma, M)$  the Floyd Max-Automaton over  $\Sigma, M$ .

For a max-automaton  $\mathcal{A}(\Sigma, M)$  each chain has a support; since there is a chain  $\# [s]^\#$  for any string  $s$  compatible with  $M$ , a string is accepted by  $\mathcal{A}(\Sigma, M)$  iff it is compatible with  $M$ . Also, whenever  $M$  is complete, each string is compatible with  $M$ , hence accepted by the max-automaton. It is not difficult to verify that a max-automaton is equivalent to a max-grammar as defined in [9]; thus, when  $M$  is complete both the max-automaton and the max-grammar define the universal language  $\Sigma^*$  by assigning to any string the (unique) structure compatible with the OPM.

In conclusion, given an OP alphabet, the OPM  $M$  assigns a structure to any string in  $\Sigma^*$  compatible with  $M$ ; a FA defined on the OP alphabet selects an appropriate subset within such a “universe”. In some sense this property is yet another variation of the fundamental Chomsky-Shützenberger theorem.

### 3 Logic characterization of FL

We are now ready to provide a characterization of FL in terms of a suitable Monadic Second Order (MSO) logic in the same vein as originally proposed by Büchi for regular languages and subsequently extended by Alur and Madhusudan for VPL. The essence of the approach consists in defining language properties in terms of relations between the positions of characters in the strings: first order variables are used to denote positions whereas second order ones denote subsets of positions; then, suitable constructions build an automaton from a given formula and conversely, in such a way that formula and corresponding automaton define the same language. The extension designed by [13] introduced a new basic binary predicate  $\leadsto$  in the syntax of the MSO logic,  $x \leadsto y$  representing the fact that in positions  $x$  and  $y$  two matching parentheses –named call and return, respectively in their terminology– are located. In the case of FL, however, we have to face new problems.

- Both finite state automata and VPA are real-time machines, i.e., they read one input character at every move; this is not the case with more general machines such as FA, which do not advance the input head when performing flush transitions, and may also apply many flush transitions before the next push or mark which are the transitions that consume input. As a consequence, whereas in the logic characterization of regular and VP languages any first order variable can belong to only one second order variable representing an automaton state, in this case –when the automaton performs a flush– the same position may correspond to different states and therefore belong to different second-order variables.
- In VPL the  $\leadsto$  relation is one-to-one, since any call matches with only one return, if any, and conversely. In FL, instead the same position  $y$  can be “paired” with different positions  $x$  in correspondence of many flush transitions with no push/mark in between, as it happens for instance when parsing a derivation such as  $A \Rightarrow^* \alpha^k A$ , consisting of  $k$  immediate derivations  $A \Rightarrow \alpha A$ ; symmetrically the same position  $x$  can be paired with many positions  $y$ .

In essence our goal is to formalize in terms of MSO formulas a complete parsing algorithm for FL, a much more complex algorithm than it is needed for regular and VP languages. The first step to achieve our goal is to define a new relation between (first order variables denoting) the positions in a string.

In some sense the new relation formalizes structural properties of FL strings in the same way as the VPL  $\leadsto$  relation does for VPL; the new relation, however, is more complex as its VPL counterpart in a parallel way as FL are much richer than VPL.

**Definition 7.** Consider a string  $s \in \Sigma^*$  and a OPM  $M$ . For  $0 \leq x < y \leq |s| + 1$ , we write  $x \leadsto y$  iff there exists a sub-string of  $\#s\#$  which is a chain  $a[r]^b$ , such that  $a$  is in position  $x$  and  $b$  is in position  $y$ .

*Example 3.* With reference to the string of Figure 1, we have  $1 \leadsto 3$ ,  $0 \leadsto 4$ ,  $6 \leadsto 8$ ,  $4 \leadsto 8$ , and  $0 \leadsto 9$ . Notice that, in the parsing of the string, such pairs correspond to contexts where a reduce operation is executed (they are listed according to their execution order).

In general  $x \leadsto y$  implies  $y > x + 1$ , and a position  $x$  may be in such a relation with more than one position and vice versa. Moreover, if  $s$  is compatible with  $M$ , then  $0 \leadsto |s| + 1$ .

### 3.1 A Monadic Second-Order Logic over Operator Precedence Alphabets

Let  $(\Sigma, M)$  be an OP alphabet. According to Definition 7 it induces the relation  $\leadsto$  over positions of characters in any words in  $\Sigma^*$ . Let us define a countable infinite set of first-order variables  $x, y, \dots$  and a countable infinite set of monadic second-order (set) variables  $X, Y, \dots$ .

**Definition 8.** The  $MSO_{\Sigma, M}$  (monadic second-order logic over  $(\Sigma, M)$ ) is defined by the following syntax:

$$\varphi := a(x) \mid x \in X \mid x \leq y \mid x \leadsto y \mid x = y + 1 \mid \neg \varphi \mid \varphi \vee \varphi \mid \exists x. \varphi \mid \exists X. \varphi$$

where  $a \in \Sigma$ ,  $x, y$  are first-order variables and  $X$  is a set variable.

$MSO_{\Sigma, M}$  formulae are interpreted over  $(\Sigma, M)$  strings and the positions of their characters in the following natural way:

- first-order variables are interpreted over positions of the string;
- second-order variables are interpreted over sets of positions;
- $a(x)$  is true iff the character in position  $x$  is  $a$ ;
- $x \leadsto y$  is true iff  $x$  and  $y$  satisfy Definition 7;
- the other logical symbols have the usual meaning.

A sentence is a formula without free variables. The language of all strings  $s \in \Sigma^*$  such that  $\#s\# \models \varphi$  is denoted by  $L(\varphi)$ :

$$L(\varphi) = \{s \in \Sigma^* \mid \#s\# \models \varphi\}$$

where  $\models$  is the standard satisfaction relation.



*Example 4.* Consider the language of Example 1, with the structure implied by its OPM. The following sentence defines it:

$$\forall z \left( \left( \begin{array}{c} call_a(z) \vee ret_a(z) \\ \vee \\ call_b(z) \vee ret_b(z) \end{array} \right) \Rightarrow \exists x, y \left( \begin{array}{c} x \curvearrowright y \wedge x < z < y \\ \wedge \\ hnd(x+1) \wedge rst(y-1) \end{array} \right) \right).$$

*Example 5.* Consider again Example 1. If we want to add the additional constraint that procedure  $b$  cannot directly install handlers (e.g. for security reasons), we may state it through the following formula:

$$\forall z (hnd(z) \Rightarrow \neg \exists u (call_b(u) \wedge (u+1 = z \vee u \curvearrowright z)))$$

We are now ready for the main result.

**Theorem 1.** *A language  $L$  over  $(\Sigma, M)$  is a FL if and only if there exists a  $MSO_{\Sigma, M}$  sentence  $\varphi$  such that  $L = L(\varphi)$ .*

The proof is constructive and structured in the following two subsections.

### 3.2 From $MSO_{\Sigma, M}$ to Floyd automata

**Proposition 1.** *Let  $(\Sigma, M)$  be an operator precedence alphabet and  $\varphi$  be a  $MSO_{\Sigma, M}$  sentence. Then  $L(\varphi)$  can be recognized by a Floyd automaton over  $(\Sigma, M)$ .*

*Proof.* The proof follows the one by Thomas [8] and is composed of two steps: first the formula is rewritten so that no predicate symbols nor first order variables are used; then an equivalent FA is built inductively.

Let  $\Sigma$  be  $\{a_1, a_2, \dots, a_n\}$ . For each predicate symbol  $a_i$  we introduce a fresh set variable  $X_i$ , therefore formula  $a_i(x)$  will be translated into  $x \in X_i$ . Following the standard construction of [8], we also translate every first order variable into a fresh second order variable with the additional constraint that the set it represents contain exactly one position.

Let  $\varphi'$  be the formula obtained from  $\varphi$  by such a translation, and consider any subformula  $\psi$  of  $\varphi'$ : let  $X_1, X_2, \dots, X_n, X_{n+1}, \dots, X_{n+m(\psi)}$  be the (second order) variables appearing in  $\psi$ . Recall that  $X_1, \dots, X_n$  represent symbols in  $\Sigma$ , hence they are never quantified.

As usual we interpret formulae over strings; in this case we use the alphabet

$$\Lambda(\psi) = \left\{ \alpha \in \{0, 1\}^{n+m(\psi)} \mid \exists! i \text{ s.t. } 1 \leq i \leq n, \alpha_i = 1 \right\}$$

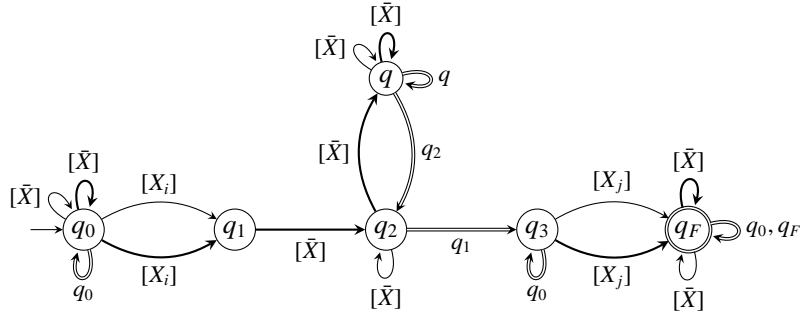
A string  $w \in \Lambda(\psi)^*$ , with  $|w| = \ell$ , is used to interpret  $\psi$  in the following way: the projection over  $j$ -th component of  $\Lambda(\psi)$  gives an evaluation  $\{1, 2, \dots, \ell\} \rightarrow \{0, 1\}$  of  $X_j$ , for every  $1 \leq j \leq n + m(\psi)$ .

For any  $\alpha \in \Lambda(\psi)$ , the projection of  $\alpha$  over the first  $n$  components encodes a symbol in  $\Sigma$ , denoted as  $symb(\alpha)$ . The matrix  $M$  over  $\Sigma$  can be naturally extended to the OPM  $M(\psi)$  over  $\Lambda(\psi)$  by defining  $M(\psi)_{\alpha, \beta} = M_{symb(\alpha), symb(\beta)}$  for any  $\alpha, \beta \in \Lambda(\psi)$ .

We now build a FA  $\mathcal{A}$  equivalent to  $\varphi'$ . The construction is inductive on the structure of the formula: first we define the FA for all atomic formulae. We give here only the

construction for  $\leadsto$ , since for the other ones the construction is standard and is the same as in [8].

Figure 2 represents the Floyd automaton for atomic formula  $\psi = X_i \leadsto X_j$  (notice that  $i, j > n$ ). For the sake of brevity, we use notation  $[X_i]$  to represent the set of all tuples  $\Lambda(\psi)$  having the  $i$ -th component equal to 1; notation  $[\bar{X}]$  represents the set of all tuples in  $\Lambda(\psi)$  having both  $i$ -th and  $j$ -th components equal to 0. The automaton, after a generic sequence of moves corresponding to visiting an irrelevant portion of the syntax tree, when reading  $X_i$  performs either a mark or a push move, depending on whether  $X_i$  is a leftmost leaf of the tree or not; then it visits the subsequent subtree ending with a flush labeled  $q_1$ ; at this point, if it reads  $X_j$ , it accepts anything else will follow the examined fragment.



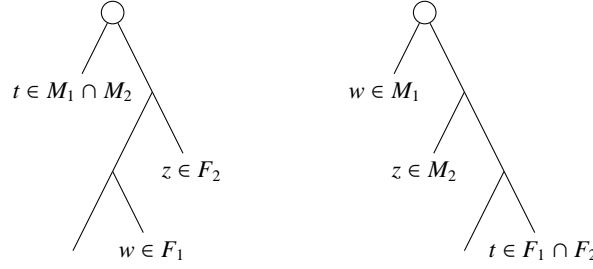
**Fig. 2.** Floyd automaton for atomic formula  $\psi = X_i \leadsto X_j$

Then, a natural inductive path leads to the construction of the automaton associated with a generic MSO formula: the disjunction of two subformulae can be obtained by building the union automaton of the two corresponding automata; similarly for negation. The existential quantification of  $X_i$  is obtained by projection erasing the  $i$ -th component. Notice that all matrices  $M(\psi)$  are well defined for any  $\psi$  because the first  $n$  components of the alphabet are never erased by quantification. The alphabet of the automaton equivalent to  $\varphi'$  is  $\Lambda(\varphi') = \{0, 1\}^n$ , which is in bijection with  $\Sigma$ .

### 3.3 From Floyd automata to $\text{MSO}_{\Sigma, M}$

Let  $\mathcal{A}$  be a deterministic Floyd automaton over  $(\Sigma, M)$ . We build a  $\text{MSO}_{\Sigma, M}$  sentence  $\varphi$  such that  $L(\mathcal{A}) = L(\varphi)$ . The main idea for encoding the behavior of the Floyd automaton is based on assigning the states visited during its run to positions along the same lines stated by Büchi [8] and extended for VPL [13]. Unlike finite state automata and VPA, however, Floyd automata do not work on-line. Hence, it is not possible to assign a single state to every position. Let  $Q = \{q_0, q_1, \dots, q_N\}$  be the states of  $\mathcal{A}$  with  $q_0$  initial; as usual, we will use second order variables to encode them. We shall need three different sets of second order variables, namely  $P_0, P_1, \dots, P_N$ ,  $M_0, M_1, \dots, M_N$

and  $F_0, F_1, \dots, F_N$ : set  $P_i$  contains those positions of  $s$  where state  $i$  may be assumed *after* a push transition.  $M_i$  and  $F_i$  represent the state reached after a flush:  $F_i$  contains the positions where the flush occurs, whereas  $M_i$  contains the positions preceding the corresponding mark. Notice that any position belongs to one only  $P_i$ , whereas it may belong to several  $F_i$  or  $M_i$  (see Figure 3).



**Fig. 3.** Example trees with a position  $t$  belonging to more than one  $M_i$  (left) and  $F_i$  (right).

We show that  $\mathcal{A}$  accepts a string  $s$  iff  $\#s\# \models \varphi$ , where

$$\begin{aligned} \varphi &:= \exists P_0, P_1, \dots, P_N, M_0, M_1, \dots, M_N, F_0, F_1, \dots, F_N, e \quad \varphi' \\ \varphi' &:= 0 \in P_0 \wedge \bigvee_{i \in F} e \in F_i \wedge \neg \exists x (e + 1 < x) \wedge \#(e + 1) \wedge \varphi_\delta \wedge \varphi_{\text{exist}} \wedge \varphi_{\text{unique}}. \end{aligned} \quad (3)$$

The first clause in  $\varphi'$  encodes the initial state, whereas the second, third and fourth ones encode the final states. We use variable  $e$  to refer to the *end* of  $s$ , i.e.,  $e$  equals the last position  $|s|$ . The remaining clauses are defined in the following: the fourth one encodes the transition function; the last ones together encode the fact that there exists exactly one state that may be assumed by a push transition in any position, and the correspondence between mark and flush transitions.

For convenience we introduce in formulae precedence relations and other shortcut notations, presented next.

*Notation.* In the following, when considering a chain  $^a[s]^b$  we assume  $s = s_0 c_1 s_1 \dots c_\ell s_\ell$ , with  $^a[c_1 c_2 \dots c_\ell]^b$  a simple chain (any  $s_g$  may be empty). Also let  $x_g$  be the position of symbol  $c_g$ , for  $g = 1, 2, \dots, \ell$  and, for the sake of uniformity, set  $c_0 = a$ ,  $x_0 = 0$ ,  $c_{\ell+1} = b$ , and  $x_{\ell+1} = |s| + 1$ .

$$\begin{aligned}
x \circ y &:= \bigvee_{M_{a,b}=\circ} a(x) \wedge b(y), \text{ for } \circ \in \{<, \doteq, >\} \\
\text{Tree}(x, z, w, y) &:= \left( \begin{array}{c} x \curvearrowright y \\ \wedge \\ (x + 1 = z \vee x \curvearrowright z) \wedge \neg \exists t (x < t < z \wedge x \curvearrowright t) \\ \wedge \\ (w + 1 = y \vee w \curvearrowright y) \wedge \neg \exists t (w < t < y \wedge w \curvearrowright t) \end{array} \right) \\
\text{Succ}_k(x, y) &:= x + 1 = y \wedge x \in P_k \\
\text{Next}_k(x, y) &:= x \curvearrowright y \wedge x \in M_k \wedge y - 1 \in F_k \\
\text{Flush}_k(x, y) &:= x \curvearrowright y \wedge x \in M_k \wedge y - 1 \in F_k \wedge \\
&\quad \left( \exists z, w \left( \text{Tree}(x, z, w, y) \wedge \bigvee_{i=0}^N \bigvee_{j=0}^N \left( \begin{array}{c} \delta(q_i, q_j) = q_k \\ \wedge \\ (\text{Succ}_i(w, y) \vee \text{Next}_i(w, y)) \\ \wedge \\ (\text{Succ}_j(x, z) \vee \text{Next}_j(x, z)) \end{array} \right) \right) \right) \\
\text{Tree}_{i,j}(x, z, w, y) &:= \text{Tree}(x, z, w, y) \wedge \left( \begin{array}{c} (\text{Succ}_i(w, y) \vee \text{Flush}_i(w, y)) \\ \wedge \\ (\text{Succ}_j(x, z) \vee \text{Flush}_j(x, z)) \end{array} \right)
\end{aligned}$$

*Remarks.* If  $x \curvearrowright y$  then there exist (unique)  $z$  and  $w$  such that  $\text{Tree}(x, z, w, y)$  holds. In particular, if  $^a[s]^b$  is a simple chain, then  $0 \curvearrowright \ell + 1$  and  $\text{Tree}(0, 1, \ell, \ell + 1)$  holds; if  $^a[s]^b$  is a composed chain, then  $0 \curvearrowright |s| + 1$  and  $\text{Tree}(0, x_1, x_\ell, x_{\ell+1})$  holds. If  $s_0 = \epsilon$  then  $x_1 = 1$ , and if  $s_\ell = \epsilon$  then  $x_\ell = |s|$ .

By definition,  $\text{Tree}_{i,j}(x, z, w, y) \wedge q_k = \delta(q_i, q_j)$  implies  $\text{Flush}_k(x, y)$ .

If  $^a[c_1 c_2 \dots c_\ell]^b$  is a simple chain with support

$$q_i = q_{t_0} \xrightarrow{c_1} q_{t_1} \xrightarrow{c_2} \dots \xrightarrow{c_\ell} q_{t_\ell} \xRightarrow{q_{t_0}} q_k \quad (4)$$

then  $\text{Tree}_{t_0, t_\ell}(0, 1, \ell, \ell + 1)$  and  $\text{Flush}_k(0, \ell + 1)$  hold; if  $^a[s_0 c_1 s_1 c_2 \dots c_\ell s_\ell]^b$  is a composed chain with support

$$q_i = q_{t_0} \xrightarrow{s_0} q_{f_0} \xrightarrow{c_1} q_{t_1} \xrightarrow{s_1} q_{f_1} \xrightarrow{c_2} \dots \xrightarrow{c_g} q_{t_g} \xrightarrow{s_g} q_{f_g} \dots \xrightarrow{c_\ell} q_{t_\ell} \xrightarrow{s_\ell} q_{f_\ell} \xRightarrow{q_{f_0}} q_k \quad (5)$$

then by induction we can see that  $\text{Tree}_{f_\ell, f_0}(0, |s_0| + 1, |s_0 \dots c_\ell|, |s| + 1)$  and  $\text{Flush}_k(0, |s| + 1)$  hold.

Formula  $\varphi_\delta$  is the conjunction of the following formulae, organized in *forward* formulae and *backward* formulae:

*Forward formulae.*

$$\varphi_{push\_fw} := \forall x, y \bigwedge_{i=0}^N \left( \begin{array}{c} (x < y \vee x \doteq y) \wedge a(y) \\ \wedge \\ \text{Succ}_i(x, y) \vee \text{Flush}_i(x, y) \end{array} \Rightarrow y \in P_{\delta(q_i, a)} \right)$$

$$\varphi_{flush\_fw} := \forall x, z, w, y \bigwedge_{i=0}^N \bigwedge_{j=0}^N \left( \text{Tree}_{i,j}(x, z, w, y) \Rightarrow \begin{array}{c} x \in M_{\delta(q_i, q_j)} \\ \wedge \\ y - 1 \in F_{\delta(q_i, q_j)} \end{array} \right)$$

Backward formulae.

$$\varphi_{push\_bw1} := \forall x, y \bigwedge_{k=0}^N \left( \begin{array}{c} (x \leq y \vee x \doteq y) \wedge a(y) \\ \wedge \\ y \in P_k \wedge x + 1 = y \end{array} \Rightarrow \bigvee_{i=0}^N (\text{Succ}_i(x, y) \wedge \delta(q_i, a) = q_k) \right)$$

$$\varphi_{push\_bw2} := \forall x, y \bigwedge_{k=0}^N \left( \begin{array}{c} (x \leq y \vee x \doteq y) \wedge a(y) \\ \wedge \\ y \in P_k \wedge x \curvearrowright y \end{array} \Rightarrow \bigvee_{i=0}^N (\text{Flush}_i(x, y) \wedge \delta(q_i, a) = q_k) \right)$$

$$\varphi_{flush\_bwM} := \forall x \bigwedge_{k=0}^N \left( x \in M_k \Rightarrow \exists y, z, w \bigvee_{i=0}^N \bigvee_{j=0}^N \left( \begin{array}{c} \text{Tree}_{i,j}(x, z, w, y) \\ \wedge \\ \delta(q_i, q_j) = q_k \end{array} \right) \right)$$

$$\varphi_{flush\_bwF} := \forall y \bigwedge_{k=0}^N \left( y \in F_k \Rightarrow \exists x, z, w \bigvee_{i=0}^N \bigvee_{j=0}^N \left( \begin{array}{c} \text{Tree}_{i,j}(x, z, w, y) \\ \wedge \\ \delta(q_i, q_j) = q_k \end{array} \right) \right)$$

$$\varphi_{flush\_bw} := \forall x, z, w, y \bigwedge_{k=0}^N \bigwedge_{i=0}^N \bigwedge_{j=0}^N \left( \begin{array}{c} \text{Tree}_{i,j}(x, z, w, y) \\ \wedge \\ \text{Flush}_k(x, y) \end{array} \Rightarrow \delta(q_i, q_j) = q_k \right)$$

Formula  $\varphi_{exist}$  is the conjunction of the following formulae:

$$\varphi_{push\_exist} := \forall x \left( \bigvee_{i=0}^N x \in P_i \right)$$

$$\varphi_{flush\_exist} := \forall x, y \left( x \curvearrowright y \Rightarrow \left( \bigvee_{k=0}^N \text{Flush}_k(x, y) \right) \right)$$

Formula  $\varphi_{unique}$  is the conjunction of the following formulae:

$$\varphi_{push\_unique} := \forall x \bigwedge_{i=0}^N \left( x \in P_i \Rightarrow \neg \bigvee_{j=0}^N (j \neq i \wedge x \in P_j) \right)$$

$$\varphi_{flush\_unique} := \forall x, y \bigwedge_{k=0}^N \left( \text{Flush}_k(x, y) \Rightarrow \neg \bigvee_{j=0}^N (j \neq k \wedge \text{Flush}_j(x, y)) \right)$$

*Remark 1.* If (3) holds, then for each  $x, y$   $\text{Succ}_i(x, y) \vee \text{Flush}_i(x, y)$  implies that such  $i$  is unique. Indeed,  $\text{Succ}_j(x, y)$  and  $\text{Flush}_k(x, y)$  are mutually exclusive; if  $\text{Flush}_i(x, y)$  then such  $i$  is unique by  $\varphi_{flush\_unique}$ ; if  $\text{Succ}_i(x, y)$  then  $y = x + 1$  and  $x \in P_i$ , thus such  $i$  is unique by  $\varphi_{push\_unique}$ .

Now let  $C = {}^a[s]^b$  be a chain in  $(\Sigma, M)$  and set

$$\psi_{i,k} := \begin{array}{c} \exists P_0, P_1, \dots, P_N \\ \exists M_0, M_1, \dots, M_N \exists e \left( 0 \in P_i \wedge \text{Flush}_k(0, e+1) \wedge \varphi_\delta \wedge \varphi_{\text{exist}} \wedge \varphi_{\text{unique}} \right) \\ \exists F_0, F_1, \dots, F_N \end{array}$$

The following lemmata hold.

**Lemma 1.** *If there exists a support  $q_i \xrightarrow{s} q_k$  for the chain  $C$  in  $\mathcal{A}$ , then  $\text{asb} \models \psi_{i,k}$ .*

*Proof.* We prove the lemma by induction on the structure of chains.

**Base step** Let  $C$  be a simple chain and its support be decomposed as in (4).

Define  $e = \ell$ , and  $P_0, P_1, \dots, P_N, M_0, \dots, M_N, F_0, \dots, F_N$  as follows.  $M_h$  is empty except for  $M_k = \{0\}$ ;  $F_h$  is empty except for  $F_k = \{\ell\}$ ; for every  $x = 0 \dots \ell$ , let  $P_h$  contain  $x$  iff  $t_x = h$  (i.e.,  $x \in P_{t_x}$ ); finally let  $P_{\delta(q_k, b)}$  contain  $\ell + 1$  if  $a < b$  or  $a \doteq b$ .

Then we show that  $\psi_{i,k}$  is satisfied by checking every subformula in  $\varphi_\delta, \varphi_{\text{exist}}, \varphi_{\text{unique}}$ .

1.  $\varphi_{\text{push\_fw}}$  is satisfied  $\forall x = y - 1 < \ell$  with  $y \in P_{\delta(q_x, a)} \wedge a(y)$ . Then  $\delta(q_t, q_{t_0}) = q_k$  guarantees  $\text{Flush}_k(0, \ell + 1)$ ; and  $\delta(q_k, b) = q_{t_{\ell+1}}$  guarantees  $\ell + 1 \in P_{\ell+1}$ .  
*Remark.* Even if  $\mathcal{A}$  is deterministic, some chains could have different supports. However, every support produces exactly one assignment  $P_{t_0}, P_{t_1}, \dots, F_k, M_k$  that satisfies  $\psi_{t_0, k}$ .
2.  $\varphi_{\text{flush\_fw}}$  is satisfied for  $x = 0, z = 1, w = \ell, y = \ell + 1$  with  $P_{t_\ell}, P_{t_0}, F_k, M_k$  (for all other cases, it is  $\neg \text{Tree}_{i,j}(x, z, w, y)$ ).
3.  $\varphi_{\text{push\_bw1}}$  is satisfied in the natural way for every  $y \leq \ell$ ; for  $y = \ell + 1$ , it is  $x > y$ ,  $x + 1 = y$ , which implies  $\neg(x < y \vee x \doteq y)$  and the antecedent is false.
4.  $\varphi_{\text{push\_bw2}}$ , for every pair  $(x, y) \neq (0, \ell + 1)$  is satisfied with  $\neg x \curvearrowright y$ ; for  $x = 0$ ,  $y = \ell + 1$ , if  $x > y$  the antecedent is false, otherwise it is satisfied with  $\text{Flush}_k(0, \ell + 1)$ ,  $P_{t_{\ell+1}}$ .
5.  $\varphi_{\text{flush\_bwM}}$  and  $\varphi_{\text{flush\_bwF}}$  are satisfied with  $x = 0$  and  $y = \ell + 1$ , respectively. (For  $x > 0, y \leq \ell$  the antecedents are false.)
6.  $\varphi_{\text{flush\_bw}}$  is satisfied in a vacuous way (false antecedent) for  $(x, y) \neq (0, \ell + 1)$ . For  $x = 0, y = \ell + 1$  it is satisfied with  $i = t_\ell, j = t_0, F_k$ .
7.  $\varphi_{\text{push\_exist}}, \varphi_{\text{push\_unique}}, \varphi_{\text{flush\_exist}}$ , and  $\varphi_{\text{flush\_unique}}$  are always satisfied, because a) the chain has a support, b)  $\mathcal{A}$  is deterministic.
8.  $\psi_{t_0, k}$  is finally satisfied with  $\text{Flush}_k(0, \ell + 1)$ .

### Induction step

Let now  $C$  be a composed chain and let its support be decomposed as in (5). Let us consider the case  $s_0 \neq \epsilon \neq s_\ell$  (other cases are similar and simpler, therefore omitted). Thus,  $\delta(q_{f_\ell}, q_{f_0}) = q_k$ .

Let  $e$  be  $|s|$ . By the inductive hypothesis, for every  $g = 0, 1, \dots, \ell$  such that  $s_g \neq \epsilon$  we have  $c_g s_g c_{g+1} \models \psi_{t_g, f_g}$ : let  $P_0^g, \dots, P_N^g, M_0^g, \dots, M_N^g, F_0^g, \dots, F_N^g$  be (the naturally shifted versions of) an assignment that satisfies  $\psi_{t_g, f_g}$ . In particular this means  $x_g \in P_{t_g} \cup M_{f_g}^g$ ,  $x_{g+1} - 1 \in F_{f_g}^g$ , and  $\text{Flush}_{f_g}(x_g, x_{g+1})$ . Then define  $P_h, M_h, F_h$  as follows. Let  $P_h$  be the union of all  $P_h^g$ ,  $M_h$  include all  $M_h^g$ ,  $F_h$  include all  $F_h^g$ . Also let  $M_k$  contain  $x_0$  and  $F_k$  contain  $x_\ell$ . Finally let  $P_{\delta(q_k, b)}$  contain  $\ell + 1$  if  $a < b$  or  $a \doteq b$ .

Then we show that  $\psi_{i,k}$  is satisfied by checking every subformula in  $\varphi_\delta, \varphi_{exist}, \varphi_{unique}$ . By the inductive hypothesis, all axioms are satisfied within every  $s_g$ . Thus, we only have to prove that they are satisfied in positions  $x_g$ , for  $0 \leq g \leq \ell$ . The proof of satisfaction of most axioms in  $\psi_{i,k}$  is clerical. Thus, we consider only a meaningful sample thereof.

1.  $\varphi_{push\_fw}$  is satisfied for  $x = x_{g-1}$  and  $y = x_g$  since  $\text{Succ}_{f_{g-1}}(x_{g-1}, x_g) \vee \text{Flush}_{f_{g-1}}(x_{g-1}, x_g)$  holds and  $\delta(q_{f_{g-1}}, c_g) = q_{t_g}, x_g \in P_{t_g}$ .
2.  $\varphi_{flush\_fw}$  is satisfied for  $\text{Tree}_{f_\ell, f_0}(0, 1, x_\ell, x_{\ell+1})$  since  $0 \in M_k, x_\ell \in F_k, \delta(q_{f_\ell}, q_{f_0}) = q_k$ .
3.  $\varphi_{push\_bw2}$  is satisfied for  $x_g \in P_{t_g}$  and  $x_{g-1} \curvearrowright x_g$  (if  $s_{g-1} \neq \epsilon$ ), since  $\text{Flush}_{f_{g-1}}(x_{g-1}, x_g)$  and  $\delta(q_{f_{g-1}}, c_g) = q_{t_g}$ .
4.  $\varphi_{push\_bwM}, \varphi_{push\_bwF}, \varphi_{push\_bw}$  are satisfied for  $\text{Tree}_{f_\ell, f_0}(0, 1, x_\ell, x_{\ell+1})$  by  $\delta(q_{f_\ell}, q_{f_0}) = q_k$ .
5.  $\varphi_{push\_unique}$ , and  $\varphi_{flush\_unique}$  are satisfied because  $\mathcal{A}$  is deterministic.

Hence  $asb \models \psi_{i,k}$ . □

**Lemma 2.** *For every chain  $C$ ,  $asb \models \psi_{i,k}$  implies that there exists a support  $q_a \xrightarrow{s} q_k$  for  $C$  in  $\mathcal{A}$ .*

*Proof.* Again, we prove the lemma by induction on the structure of chains.

**Base step** First consider the induction bases with  $s_g = \epsilon$  for every  $g = 0, 1, \dots, \ell$ , i.e.,  $^a[s]^b$  is a simple chain with  $s = c_1 c_2 \dots c_\ell$ . Let  $asb \models \psi_{i,k}$ . Hence there is a suitable assignment for  $e, P_h, M_h, F_h$  such that  $0 \in P_i \wedge \text{Flush}_k(0, e+1) \wedge \varphi_\delta \wedge \varphi_{exist} \wedge \varphi_{unique}$  holds true. Clearly  $e$  is  $|s|$ . For every  $g$ , let  $t_g$  be the index such that  $g \in P_{t_g}$ . Notice that  $t_g$  is unique by  $\varphi_{push\_unique}$  and in particular  $t_0 = i$ . Hence  $t_g$  is the unique index such that  $\text{Succ}_{t_g}(g, g+1)$ . Then, by  $\varphi_{push\_bw1}$  with  $y = g < \ell$ , we have  $\delta(q_{t_g}, c_{g+1}) = q_{t_{g+1}}$ . Moreover, since  $\text{Flush}_k(0, \ell+1) \wedge \text{Tree}_{t_\ell, t_0}(0, 1, \ell, \ell+1)$ , by  $\varphi_{flush\_bw}$  we get  $\delta(q_{t_\ell}, q_{t_0}) = q_k$ . Hence we have built a support like (4).

**Induction step** Now consider the general case with  $s = s_0 c_1 s_1 \dots c_\ell s_\ell$  and again consider the assignment for  $P_h, M_h, F_h$  that satisfies  $\psi_{i,k}$ . For every  $g$ , let  $t_g$  be the index such that  $x_g \in P_{t_g}$ , and notice that  $t_g$  is unique by  $\varphi_{push\_unique}$ ; in particular  $t_0 = i$ . For  $g = 0, 1, \dots, \ell$ , since  $x_g \curvearrowright x_{g+1} \vee x_{g+1} = x_g + 1$ , let  $f_g$  be the index such that  $\text{Flush}_{f_g}(x_g, x_{g+1}) \vee \text{Succ}_{f_g}(x_g, x_{g+1})$ . Notice that such  $f_g$  is unique by  $\varphi_{unique}$  (see Remark 1), moreover  $s_g = \epsilon$  implies  $f_g = t_g$ . Hence if  $s_g \neq \epsilon$ , we have  $c_g s_g c_{g+1} \models \psi_{t_g, f_g}$  and, by the inductive hypothesis, there exists a support  $q_{t_g} \xrightarrow{s_g} q_{f_g}$  in  $\mathcal{A}$ .

For every  $g = 0 < \ell$ , since  $f_g$  is unique, by applying  $\varphi_{push\_bw1}$  with  $y = x_{g+1}$  we get  $\delta(q_{f_g}, c_{g+1}) = q_{t_{g+1}}$ . Moreover, since  $\text{Tree}_{i, t_\ell}(x_0, x_1, x_\ell, x_{\ell+1}) \wedge \text{Flush}_k(x_0, x_{\ell+1})$ , by  $\varphi_{flush\_bw}$  we get  $\delta(q_{t_\ell}, q_i) = q_k$ . Hence we have built a support like (5) and this concludes the proof. □

**Proposition 2.** *Let  $(\Sigma, M)$  be an operator precedence alphabet and  $\mathcal{A}$  be a Floyd automaton over  $(\Sigma, M)$ . Then there exists an MSO $_{\Sigma, M}$  sentence  $\varphi$  such that  $L(\mathcal{A}) = L(\varphi)$ .*

*Proof.* Let  $\varphi$  be the MSO $_{\Sigma, M}$  sentence defined in (3). We show that  $L(\mathcal{A}) = L(\varphi)$  by applying the previous lemmata. Consider an accepting computation of  $s$  in  $\mathcal{A}$ . Then there

exists a support  $q_0 \xrightarrow{s} q_k$  for the chain  $\# [s]^\#$ , with  $q_k$  a final state; hence by Lemma 1,  $\#s\# \models \psi_{0,k}$ . Vice versa, let  $s \in L(\varphi)$ , then  $\#s\# \models \psi_{0,k}$  with  $q_k$  a final state; hence Lemma 2 implies that there exists a path  $q_0 \xrightarrow{s} q_j$  and this concludes the proof.  $\square$

## 4 Conclusions and future work

This paper somewhat completes a research path that began more than four decades ago and was resumed only recently with new -and old- goals. FL enjoy most of the nice properties that made regular languages highly appreciated and applied to achieve decidability and, therefore, automatic analysis techniques. In this paper we added to the above collection the ability to formalize and analyze FL by means of suitable MSO logic formulae.

New research topics, however, stimulate further investigation. Here we briefly mention only two mutually related ones. On the one hand, FA devoted to analyze strings should be extended in the usual way into suitable transducers. They could be applied, e.g. to translate typical mark-up languages such as XML, HTML, Latex, ... into their end-user view. Such languages, which motivated also the definition of VPL, could be classified as “explicit parenthesis languages” (EPL), i.e. languages whose syntactic structure is explicitly apparent in the input string. On the other hand, we plan to start from the remark that VPL are characterized by a well precise shape of the OPM [2] to characterize more general classes of such EPL: for instance the language of Example 1 is such a language that is not a VPL, however. Another notable feature of FL, in fact, is that they are suitable as well to parse languages with implicit syntax structure such as most programming languages as to analyze and translate EPL.

## References

1. Floyd, R.W.: Syntactic analysis and operator precedence. *Journ. ACM* **10** (1963) 316–333
2. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. *Journal of Computer and System Science* (2012) to appear.
3. Grune, D., Jacobs, C.J.: *Parsing techniques: a practical guide*. Springer, New York (2008)
4. Berstel, J., Boasson, L.: Balanced grammars and their languages. In et al., W.B., ed.: *Formal and Natural Computing*. Volume 2300 of LNCS., Springer (2002) 3–25
5. Nowotka, D., Srba, J.: Height-deterministic pushdown automata. In Kucera, L., Kucera, A., eds.: *MFCS 2007, Český Krumlov, Czech Republic, August 26-31, 2007, Proceedings*. Volume 4708 of LNCS., Springer (2007) 125–134
6. Caucal, D.: Boolean algebras of unambiguous context-free languages. In Hariharan, R., Mukund, M., Vinay, V., eds.: *FSTTCS 2008, Dagstuhl, Germany* (2008)
7. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: *STOC: ACM Symposium on Theory of Computing (STOC)*. (2004)
8. Thomas, W.: Automata on infinite objects. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. (1990) 133–192
9. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. *Information and Control* **37** (1978) 115–133
10. Barengi, A., Crespi Reghizzi, S., Mandrioli, D., Ponte, V., Pradella, M., Viviani, E.: Practical parallel parsing for large texts. Submitted for publication (2012)



11. Lonati, V., Mandrioli, D., Pradella, M.: Precedence automata and languages. In Kulikov, A.S., Vereshchagin, N.K., eds.: CSR. Volume 6651 of Lecture Notes in Computer Science., Springer (2011) 291–304
12. Lonati, V., Mandrioli, D., Pradella, M.: Precedence automata and languages. CoRR **abs/1012.2321** (2010)
13. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM **56** (2009)
14. Fischer, M.J.: Some properties of precedence languages. In: STOC '69: Proc. first annual ACM Symp. on Theory of Computing, New York, NY, USA, ACM (1969) 181–190